Lab Code [10 points]
Filename: grader_test.sv
AndrewID: tbeasley

```
`default_nettype none

module Grader_test; // Test Grader module and Grader_FMS
  logic [11:0] Guess, Guess_pos, masterPattern;
  logic Grade_it_L, Gclr, Gload, CLOCK_50, reset_L;
  logic AeqB1, AeqB2, AeqB3, AeqB4;
  logic [3:0] Znarly, Zood, sum1, sum2, sum3, sum4, sum5, sum6, sum7;
  logic [2:0] Tshape, Cshape, Oshape, Dshape, Ishape, Zshape;
  logic [3:0] T_count, C_count, O_count, D_count, I_count, Z_count;
  logic [2:0] partial_Zoods;

  Grader g1(.*);
  Grader_FSM f1(.*);

  initial begin
    CLOCK_50 = 0;
    reset_L <= 0;
    forever #5 CLOCK_50 = ~CLOCK_50;
  end

  initial begin
    $monitor("masterPattern = %b | Guess = %b | %d Znarly(s) | %d Zood(s)",
              masterPattern, Guess, Znarly, Zood);

  // Moore Machine, will output correct value on next clock edge

  /* T = 001
     C = 010
     O = 011
     D = 100
     I = 101
     Z = 110 */

    @(posedge CLOCK_50); // INIT, no value
    reset_L <= 1; // ignore reset
    Grade_it_L <=1;
    @(posedge CLOCK_50); // INIT, no value


    Grade_it_L <= 0;
    @(posedge CLOCK_50); // SAVE, no value
    @(posedge CLOCK_50); // SAVE, no value

    Grade_it_L <= 1;
    @(posedge CLOCK_50); // HOLD, no value

    $display("//masterPattern 1//");

    masterPattern <= 12'b101_110_100_001; // IZDT
    Guess <= 12'b001_001_010_010; // TTCC
    @(posedge CLOCK_50); // HOLD, 0 Znarlys, 1 Zood

    Guess <= 12'b011_011_100_100; // OODD
    @(posedge CLOCK_50); // HOLD, 1 Znarly, 0 Zoods

    Guess <= 12'b101_101_010_010; // IICC
    @(posedge CLOCK_50); // HOLD, 1 Znarly, 0 Zoods

    Guess <= 12'b101_011_001_110; // IOTZ
    @(posedge CLOCK_50); // HOLD, 1 Znarly, 2 Zoods

    Guess <= 12'b001_101_110_100; // TIZD
    @(posedge CLOCK_50); // HOLD, 0 Znarlys, 4 Zoods

    Guess <= 12'b101_110_100_001; // IZDT
    @(posedge CLOCK_50); // HOLD, 4 Znarlys, 0 Zoods

    Grade_it_L <= 0;
    @(posedge CLOCK_50); // SAVE, clear Register Guess value
```

```
70      @(posedge CLOCK_50);
71
72
73
74      Grade_it_L <= 1;
75      @(posedge CLOCK_50); // HOLD
76
77      $display("//masterPattern 2//");
78
79      masterPattern <= 12'b011_011_011_011; // OOOO
80      Guess <= 12'b001_001_010_010; // TTCC
81      @(posedge CLOCK_50); // HOLD, 0 Znarlys, 0 Zood
82
83      Guess <= 12'b011_011_100_100; // OODD
84      @(posedge CLOCK_50); // HOLD, 2 Znarly, 0 Zoods
85
86      Guess <= 12'b101_101_010_010; // IICC
87      @(posedge CLOCK_50); // HOLD, 0 Znarly, 0 Zoods
88
89      Guess <= 12'b101_011_001_110; // IOTZ
90      @(posedge CLOCK_50); // HOLD, 1 Znarly, 0 Zoods
91
92      Guess <= 12'b001_101_110_100; // TIZD
93      @(posedge CLOCK_50); // HOLD, 0 Znarlys, 0 Zoods
94
95      Guess <= 12'b011_011_011_011; // 0000
96      @(posedge CLOCK_50); // HOLD, 4 Znarlys, 0 Zoods
97
98      Grade_it_L <= 0;
99      @(posedge CLOCK_50); // SAVE, clear Register Guess value
100     @(posedge CLOCK_50);
101
102     #5 $finish;
103   end
104
105 endmodule: Grader_test
```

Lab Code [10 points]
Filename: library.sv
AndrewID: tbeasley

```
`default_nettype none

module Decoder
  #(parameter WIDTH = 8)
  (input logic [$clog2(WIDTH) - 1:0] I,
  input logic en,
  output logic [WIDTH-1:0] D);

  assign D = (en) ? (2**I) : '0;

endmodule: Decoder

module BarrelShifter
  (input logic [15:0] V,
  input logic [3:0] by,
  output logic [15:0] S);

  assign S = V << by;

endmodule: BarrelShifter


module Multiplexer
  #(parameter WIDTH = 8)
  (input logic [WIDTH-1:0] I,
  input logic [$clog2(WIDTH) - 1:0] S,
  output logic Y);

  assign Y = I[S];

endmodule: Multiplexer


module Mux2to1
  #(parameter WIDTH = 7)
  (input logic [WIDTH-1:0] I0,
  input logic [WIDTH-1:0] I1,
  input logic S,
  output logic [WIDTH-1:0] Y);

  assign Y = (S === 1'b1) ? I1 : I0;


endmodule: Mux2to1


module MagComp
  #(parameter WIDTH = 8)
  (input logic [WIDTH-1:0] A,
  input logic [WIDTH-1:0] B,
  output logic AltB,
  output logic AeqB,
  output logic AgtB);

  assign AltB = (A < B);
  assign AeqB = (A === B);
  assign AgtB = (A > B);

endmodule: MagComp


module Comparator
  #(parameter WIDTH = 4)
  (input logic [WIDTH-1:0] A,
  input logic [WIDTH-1:0] B,
  output logic AeqB);

  assign AeqB = (A === B);

```

```
 70
 71 endmodule: Comparator
 72
 73
 74 module Adder
 75   #(parameter WIDTH = 8)
 76   (input logic cin,
 77   input logic [WIDTH-1:0] A, B,
 78   output logic cout,
 79   output logic [WIDTH-1:0] sum);
 80
 81   assign {cout, sum} = A + B + cin;
 82
 83 endmodule: Adder
 84
 85 module Subtracter
 86   #(parameter WIDTH = 8)
 87   (input logic bin,
 88   input logic [WIDTH-1:0] A, B,
 89   output logic bout,
 90   output logic [WIDTH-1:0] diff);
 91
 92   assign {bout, diff} = A - B - bin;
 93
 94
 95 endmodule: Subtracter
 96
 97
 98 module DFlipFlop
 99   (input logic preset_L, D, clock, reset_L,
100   output logic Q);
101
102   always_ff @(posedge clock, negedge reset_L, negedge preset_L)
103     if (~reset_L)
104       Q <= 0;
105     else if (~preset_L)
106       Q <= 1;
107     else
108       Q <= D;
109
110 endmodule: DFlipFlop
111
112
113 module Register
114   #(parameter WIDTH = 8)
115   (input logic en, clear, clock,
116   input logic [WIDTH - 1:0] D,
117   output logic [WIDTH - 1:0] Q);
118
119   always_ff @(posedge clock)
120     if (en)
121       Q <= D;
122     else if (clear)
123       Q <= '0;
124
125 endmodule: Register
126
127
128 module Counter
129   #(parameter WIDTH = 8)
130   (input logic en, clear, load, up, clock,
131   input logic [WIDTH-1:0] D,
132   output logic [WIDTH-1:0] Q);
133
134    always_ff @(posedge clock)
135     if (clear)
136       Q <= '0;
137     else if (load)
138       Q <= D;
139     else if (up && en)
140       Q <= Q + 1'd1;
```

```
141     else if (~up && en)
142       Q <= Q - 1'd1;
143
144 endmodule: Counter
145
146
147 module Synchronizer
148   (input logic async, clock,
149   output logic sync);
150
151     logic buffer;
152
153     always_ff @(posedge clock) begin
154       buffer <= async;
155       sync <= buffer;
156     end
157
158 endmodule: Synchronizer
159
160
161 module ShiftRegisterSIPO
162   #(parameter WIDTH = 8)
163   (input logic en, left, serial, clock,
164   output logic [WIDTH-1:0] Q);
165
166   always_ff @(posedge clock)
167     if (left && en)
168       Q <= {Q[WIDTH-2:0], serial};
169     else if (~left && en)
170       Q <= {serial, Q[WIDTH-1:1]};
171
172 endmodule: ShiftRegisterSIPO
173
174
175 module ShiftRegisterPIPO
176   #(parameter WIDTH = 8)
177   (input logic en, left, load, clock,
178   input logic [WIDTH-1:0] D,
179   output logic [WIDTH-1:0] Q);
180
181   always_ff @(posedge clock)
182     if (load)
183       Q <= D;
184     else if (left && en && ~load)
185       Q <= Q << 1;
186     else if (~left && en && ~load)
187       Q <= Q >> 1;
188
189 endmodule: ShiftRegisterPIPO
190
191
192 module BarrelShiftRegister
193   #(parameter WIDTH = 8)
194   (input logic en, load, clock,
195   input logic [1:0] by,
196   input logic [WIDTH-1:0] D,
197   output logic [WIDTH-1:0] Q);
198
199   always_ff @(posedge clock)
200     if (load)
201       Q <= D;
202     else if (en)
203       Q <= Q << by;
204
205 endmodule: BarrelShiftRegister
206
207 module BusDriver
208   #(parameter WIDTH = 8)
209   (input logic en,
210   input logic [WIDTH-1:0] data,
211   output logic [WIDTH-1:0] buff,
```

```
212   inout tri [WIDTH-1:0] bus);
213
214   assign bus = (en) ? data : 'bz;
215   assign buff = bus;
216
217 endmodule: BusDriver
218
219
220 module Memory
221   #(parameter DW = 16,
222               W  = 256,
223               AW = $clog2(W))
224   (input logic re, we, clock,
225   input logic [AW-1:0] addr,
226   inout tri [DW-1:0] data);
227
228   logic [DW-1:0] M[W];
229   logic [DW-1:0] rData;
230
231   assign data = (re) ? rData: 'bz;
232
233   always_ff @(posedge clock)
234     if (we)
235       M[addr] <= data;
236
237   always_comb
238     rData = M[addr];
239
240 endmodule: Memory
```

Lab Code [10 points]
Filename: mastermindVGA.sv
AndrewID: tbeasley

```
/** FILE
 *  mastermindVGA.sv
 *
 *  BRIEF
 *  Module that acts an an interface between the Mastermind game and
 *  the VGA output.
 *
 *  The game field will look like a standard Mastermind playing field,
 *  with a small number in the lower right indicating the number of games
 *  available.
 *
 *  Zorgian terminology:
 *  Znarly = correct shape, correct spot
 *  Zood   = correct shape, wrong spot
 *
 *  AUTHOR
 *  Anita Zhang (anitazha)
 */

/****** File-wide Colors ***********/
typedef enum logic [23:0] {
    RED      = {8'hFF, 8'h00, 8'h00},
    GREEN    = {8'h00, 8'hFF, 8'h00},
    BLUE     = {8'h00, 8'h00, 8'hFF},
    CYAN     = {8'h00, 8'hFF, 8'hCC},
    PURPLE   = {8'h99, 8'h00, 8'hFF},
    YELLOW   = {8'hFF, 8'hFF, 8'h00},
    BLACK    = {8'h00, 8'h00, 8'h00},
    WHITE    = {8'hFF, 8'hFF, 8'hFF}
} color_t;

/****** File-wide Shapes ***********/
typedef enum logic [2:0] {
    LEFTTOP  = 3'b001,  // blue
    WALL     = 3'b010,  // red
    RIGHTTOP = 3'b011,  // cyan
    EQUAL    = 3'b100,  // purple
    RIGHTBOT = 3'b101,  // green
    LEFTBOT  = 3'b110   // yellow
} shape_t;

/** BRIEF
 *  Main module that handles user input and displays game data.
 */

module mastermindVGA (
    input  logic        CLOCK_50,
    // VGA display signals -- route directly to FPGA pins
    output logic [7:0]  VGA_R, VGA_G, VGA_B,
    output logic        VGA_BLANK_N, VGA_CLK, VGA_SYNC_N,
    output logic        VGA_VS, VGA_HS,
    // game information
    input  logic [3:0]  numGames,
    input  logic        loadNumGames,
    // Items for a particular round
    input  logic [3:0]  roundNumber,
    input  logic [11:0] guess,
    input  logic        loadGuess,
    input  logic [3:0]  znarly, zood,
    input  logic        loadZnarlyZood,
    input  logic        clearGame,
    // master patterns
    input  logic [11:0] masterPattern,
    input  logic        displayMasterPattern,
    // other
    input  logic        reset
    );

    /*************************************
```

```
70       *       Internal Signals
71       ****************************************/
72
73      // game data
74      logic [7:0][3:0][2:0] memGuess;
75      logic [7:0][3:0]      memZnarly;
76      logic [7:0][3:0]      memZood;
77      logic [3:0]           memNumGames;
78      logic [3:0][2:0]      master;
79      // VGA data
80      logic [9:0]           x, y;
81      logic                 blank;
82      // drawing data
83      logic [2:0]           shapeSel;
84      logic [3:0]           numValue;
85      logic [1:0]           masterIdx, guessIdxX;
86      logic [2:0]           guessIdxY;
87      logic [3:0]           gIdxX, gIdxY;
88      logic                 inGameZoneX, inGameZoneY;
89      logic                 isNum, isCredit, isRound;
90      logic                 isMaster, isZZ, isShape;
91      color_t               zzColor, shapeColor;
92      color_t               color;
93      // loop counters
94      integer               i, j;
95      // other
96      logic                 clk;
97
98      // game playing field
99      localparam X0 = 10'd169;
100     localparam X1 = 10'd481;
101     localparam Y0 = 10'd10;
102     localparam Y1 = 10'd468;
103     localparam GSIDE = 10'd52;    // grid width
104     // specific playing field coordinates
105     localparam ZZ_X = 10'd429;    // znarly/zood X position
106     localparam MASTER_Y = 10'd426; // master pattern Y position
107
108     // renamed signals
109     assign clk = CLOCK_50;
110     assign master = masterPattern;
111
112     /****************************************
113      *       VGA data
114      ****************************************/
115
116     vga vgaCounter (
117             .row        (y),
118             .col        (x),
119             .HS         (VGA_HS),
120             .VS         (VGA_VS),
121             .*);
122
123     assign VGA_BLANK_N           = ~blank;
124     assign VGA_CLK               = CLOCK_50;
125     assign VGA_SYNC_N            = 1'b0;
126     assign {VGA_R, VGA_G, VGA_B} = color;
127
128     /****************************************
129      *          Store Game Info
130      ****************************************/
131
132     registerAZ #(4) numGamesReg (
133             .Q      (memNumGames),
134             .D      (numGames),
135             .clr    (clearGame),
136             .en     (loadNumGames),
137             .*);
138
139     // have guess separate so the switch flipping is displayed
140     always_ff @(posedge clk, posedge reset) begin
```

```
141         if (reset)
142             memGuess <= 96'b0;
143         else if (clearGame)
144             memGuess <= 96'b0;
145         else if (loadGuess)
146             memGuess[roundNumber] <= guess;
147     end
148 
149     // only store znarly and zood when ready
150     always_ff @(posedge clk, posedge reset) begin
151         if (reset) begin
152             memZnarly <= 32'b0;
153             memZood <= 32'b0;
154         end
155         else if (clearGame) begin
156             memZnarly <= 32'b0;
157             memZood <= 32'b0;
158         end
159         else if (loadZnarlyZood) begin
160             memZood[roundNumber]   <= zood;
161             memZnarly[roundNumber] <= znarly;
162         end
163     end
164 
165     /**************************************
166      *          Color/Boundary Assignments
167      **************************************/
168 
169     range_check gameFieldX (
170             .val        (x),
171             .low        (X0),
172             .high       (X1),
173             .is_between (inGameZoneX)
174             );
175 
176     range_check gameFieldY (
177             .val        (y),
178             .low        (Y0),
179             .high       (Y1),
180             .is_between (inGameZoneY)
181             );
182 
183     always_comb begin
184         color = BLACK;
185         if (inGameZoneX & inGameZoneY) begin
186             // round number
187             if (isRound & isNum)
188                 color = WHITE;
189             // credits
190             else if (isCredit & isNum)
191                 color = CYAN;
192             // znarly/zood
193             else if (isZZ)
194                 color = zzColor;
195             // master shape
196             else if (isMaster & displayMasterPattern)
197                 color = shapeColor;
198             // guess shape
199             else if (isShape)
200                 color = shapeColor;
201         end
202     end
203 
204     /**************************************
205      *       Draw Things
206      **************************************/
207 
208     // grid index (of the playing field) -- "grid" is 6 x 9
209     assign gIdxX     = (inGameZoneX ? ((x - X0) / GSIDE) : 4'b1111);
210     assign gIdxY     = (inGameZoneY ? ((y - Y0) / GSIDE) : 4'b1111);
211 
```

```
212     // define "zones" for each shape type
213     assign isRound   = ((gIdxX == 4'd0) & (gIdxY < 4'd8));
214     assign isZZ      = ((gIdxX == 4'd5) & (gIdxY < 4'd8));
215     assign isCredit  = ((gIdxX == 4'd5) & (gIdxY == 4'd8));
216     assign isMaster  = ((gIdxX > 4'd0) & (gIdxX < 4'd5) & (gIdxY == 4'd8));
217     assign isShape   = ((gIdxX > 4'd0) & (gIdxX < 4'd5) & (gIdxY < 4'd8));
218
219     // indices/signals for drawing the shapes
220     assign numValue  = (isCredit ? memNumGames : gIdxY);
221     assign shapeSel  = (isMaster ? master[masterIdx] :
222                         memGuess[guessIdxY][guessIdxX]);
223     assign masterIdx = (isMaster ? ~(gIdxX - 1'b1) : 2'b0);
224     assign guessIdxX = (isShape  ? ~(gIdxX - 1'b1) : 2'b0);
225     assign guessIdxY = (isShape  ? gIdxY : 4'b0);
226
227     // draw the round numbers on the side, or the credit value
228     drawNumber numDrawer (
229             .inNum  (isNum),
230             .x      (x),
231             .y      (y),
232             .posX   (X0 + (gIdxX * GSIDE)),
233             .posY   (Y0 + (gIdxY * GSIDE)),
234             .value  (numValue)
235             );
236
237     // draw shapes for the guess field
238     drawShape shapeDrawer (
239             .color  (shapeColor),
240             .x      (x),
241             .y      (y),
242             .posX   (X0 + (gIdxX * GSIDE)),
243             .posY   (Y0 + (gIdxY * GSIDE)),
244             .shape  (shapeSel)
245             );
246
247     // draw Znarly/Zood results
248     drawZnarlyZood zzDrawer (
249             .color  (zzColor),
250             .znarly (memZnarly[gIdxY]),
251             .zood   (memZood[gIdxY]),
252             .x      (x),
253             .y      (y),
254             .posX   (ZZ_X),
255             .posY   (Y0 + (gIdxY * GSIDE))
256             );
257
258 endmodule: mastermindVGA
259
260 /*****************************************************************
261  *
262  *                    Drawing modules
263  *
264  *****************************************************************/
265
266 /** BRIEF
267  *  Given the position of a 42x42 px box (the upper left coordinate),
268  *  draw a number specified by "value". One of the inputs will be the
269  *  current (x, y) coordinate being processed, and a bit will be
270  *  output according to whether that pixel is in the number's zone
271  */
272 module drawNumber
273     #(parameter LINEWIDTH = 10'd4, PADDING = 10'd10, SIDE = 10'd42) (
274     output logic         inNum,
275     input  logic [9:0]   x, y,
276     input  logic [9:0]   posX, posY,
277     input  logic [2:0]   value
278     );
279
280     // internal signals
281     logic   [6:0] isSegX, isSegY, isSeg;
282
```

```
283     /***************************************
284      *          Output logic
285      ***************************************/
286
287     assign isSeg = (isSegX & isSegY);
288
289     always_comb begin
290         inNum = 1'b0;
291
292         case (value)
293             3'd0: begin
294                 if (isSeg[5:0])
295                     inNum = 1'b1;
296             end
297             3'd1: begin
298                 if (isSeg[2:1])
299                     inNum = 1'b1;
300             end
301             3'd2: begin
302                 if (isSeg[0] | isSeg[1] | isSeg[6] | isSeg[4] | isSeg[3])
303                     inNum = 1'b1;
304             end
305             3'd3: begin
306                 if (isSeg[3:0] || isSeg[6])
307                     inNum = 1'b1;
308             end
309             3'd4: begin
310                 if (isSeg[6:5] || isSeg[2:1])
311                     inNum = 1'b1;
312             end
313             3'd5: begin
314                 if (isSeg[6:5] || isSeg[3:2] || isSeg[0])
315                     inNum = 1'b1;
316             end
317             3'd6: begin
318                 if (isSeg[6:2] || isSeg[0])
319                     inNum = 1'b1;
320             end
321             3'd7: begin
322                 if (isSeg[2:0])
323                     inNum = 1'b1;
324             end
325         endcase
326     end
327
328     /***************************************
329      *          Segment Boundary Check
330      ***************************************/
331
332     // top segment
333     offset_check #(10) segCheckX0 (
334             .val        (x),
335             .low        (posX + PADDING),
336             .delta      (SIDE - (2*PADDING)),
337             .is_between (isSegX[0]));
338
339     offset_check #(10) segCheckY0 (
340             .val        (y),
341             .low        (posY + PADDING),
342             .delta      (LINEWIDTH),
343             .is_between (isSegY[0]));
344
345     // top right segment
346     offset_check #(10) segCheckX1 (
347             .val        (x),
348             .low        (posX + (SIDE - PADDING) - LINEWIDTH),
349             .delta      (LINEWIDTH),
350             .is_between (isSegX[1]));
351
352     offset_check #(10) segCheckY1 (
353             .val        (y),
```

```
354             .low        (posY + PADDING),
355             .delta      ((SIDE - (PADDING*2))/2),
356             .is_between (isSegY[1]));
357
358     // bottom right segment
359     offset_check #(10) segCheckX2 (
360             .val        (x),
361             .low        (posX + (SIDE - PADDING) - LINEWIDTH),
362             .delta      (LINEWIDTH),
363             .is_between (isSegX[2]));
364
365     offset_check #(10) segCheckY2 (
366             .val        (y),
367             .low        (posY + PADDING + ((SIDE - (2*PADDING))/2)),
368             .delta      ((SIDE - (PADDING*2))/2),
369             .is_between (isSegY[2]));
370
371     // bottom segment
372     offset_check #(10) segCheckX3 (
373             .val        (x),
374             .low        (posX + PADDING),
375             .delta      (SIDE - (2*PADDING)),
376             .is_between (isSegX[3]));
377
378     offset_check #(10) segCheckY3 (
379             .val        (y),
380             .low        (posY + (SIDE - PADDING) - LINEWIDTH),
381             .delta      (LINEWIDTH),
382             .is_between (isSegY[3]));
383
384     // bottom left segment
385     offset_check #(10) segCheckX4 (
386             .val        (x),
387             .low        (posX + PADDING),
388             .delta      (LINEWIDTH),
389             .is_between (isSegX[4]));
390
391     offset_check #(10) segCheckY4 (
392             .val        (y),
393             .low        (posY + PADDING + ((SIDE - (2*PADDING))/2)),
394             .delta      ((SIDE - (PADDING*2))/2),
395             .is_between (isSegY[4]));
396
397     // top left segment
398     offset_check #(10) segCheckX5 (
399             .val        (x),
400             .low        (posX + PADDING),
401             .delta      (LINEWIDTH),
402             .is_between (isSegX[5]));
403
404     offset_check #(10) segCheckY5 (
405             .val        (y),
406             .low        (posY + PADDING),
407             .delta      ((SIDE - (PADDING*2))/2),
408             .is_between (isSegY[5]));
409
410     // middle segment
411     offset_check #(10) segCheckX6 (
412             .val        (x),
413             .low        (posX + PADDING),
414             .delta      (SIDE - (2*PADDING)),
415             .is_between (isSegX[6]));
416
417     offset_check #(10) segCheckY6 (
418             .val        (y),
419             .low        (posY + (SIDE/2) - LINEWIDTH/2),
420             .delta      (LINEWIDTH),
421             .is_between (isSegY[6]));
422
423 endmodule: drawNumber
424
```

```
425 /** BRIEF
426  *  Given the position of a 42x42 px box (the upper left coordinate),
427  *  draw the znarly/zood results. One of the inputs will be the
428  *  current (x, y) coordinate being processed, and a color will be
429  *  output according to whether that pixel is in the shape's zone.
430  *
431  *  Znarly is red; zood white (because I saw it on the internet)
432  */
433 module drawZnarlyZood
434     #(parameter WIDTH = 10'd26, PADDING = 10'd16) (
435     output color_t        color,
436     input  logic   [3:0]  znarly, zood,
437     input  logic   [9:0]  x, y,
438     input  logic   [9:0]  posX, posY
439     );
440
441     /**************************************
442      *       Internal Signals
443      **************************************/
444
445     logic   [3:0] inBoxX, inBoxY;
446
447     /**************************************
448      *       Boundary Checks
449      **************************************/
450
451     // create boundaries for 4 tiny squares
452     // top left
453     offset_check #(10) squareCheckX0 (
454             .val        (x),
455             .low        (posX),
456             .delta      (PADDING),
457             .is_between (inBoxX[0]));
458
459     offset_check #(10) squareCheckY0 (
460             .val        (y),
461             .low        (posY),
462             .delta      (PADDING),
463             .is_between (inBoxY[0]));
464
465     // top right
466     offset_check #(10) squareCheckX1 (
467             .val        (x),
468             .low        (posX + WIDTH),
469             .delta      (PADDING),
470             .is_between (inBoxX[1]));
471
472     offset_check #(10) squareCheckY1 (
473             .val        (y),
474             .low        (posY),
475             .delta      (PADDING),
476             .is_between (inBoxY[1]));
477
478     // bottom left
479     offset_check #(10) squareCheckX2 (
480             .val        (x),
481             .low        (posX),
482             .delta      (PADDING),
483             .is_between (inBoxX[2]));
484
485     offset_check #(10) squareCheckY2 (
486             .val        (y),
487             .low        (posY + WIDTH),
488             .delta      (PADDING),
489             .is_between (inBoxY[2]));
490
491     // bottom right
492     offset_check #(10) squareCheckX3 (
493             .val        (x),
494             .low        (posX + WIDTH),
495             .delta      (PADDING),
```

```
496             .is_between (inBoxX[3]));
497
498     offset_check #(10) squareCheckY3 (
499             .val        (y),
500             .low        (posY + WIDTH),
501             .delta      (PADDING),
502             .is_between (inBoxY[3]));
503
504     /*************************************
505      *       Color Assignments
506      *************************************/
507
508     always_comb begin
509         color = BLACK;
510
511         if (inBoxX[0] & inBoxY[0]) begin
512             if (znarly > 3'd0)
513                 color = RED;
514             else if (zood > 3'd3)
515                 color = WHITE;
516         end
517         else if (inBoxX[1] & inBoxY[1]) begin
518             if (znarly > 3'd1)
519                 color = RED;
520             else if (zood > 3'd2)
521                 color = WHITE;
522         end
523         else if (inBoxX[2] & inBoxY[2]) begin
524             if (znarly > 3'd2)
525                 color = RED;
526             else if (zood > 3'd1)
527                 color = WHITE;
528         end
529         else if (inBoxX[3] & inBoxY[3]) begin
530             if (znarly > 3'd3)
531                 color = RED;
532             else if (zood > 3'd0)
533                 color = WHITE;
534         end
535     end
536
537 endmodule: drawZnarlyZood
538
539 /** BRIEF
540  *  Given the position of a 42x42 px box (the upper left coordinate),
541  *  draw a shape specified by "shape". One of the inputs will be the
542  *  current (x, y) coordinate being processed, and a color will be
543  *  output according to whether that pixel is in the shape's zone.
544  */
545 module drawShape
546     #(parameter LINEWIDTH = 10'd15, SIDE = 10'd42) (
547     output color_t        color,
548     input  logic   [9:0]  x, y,
549     input  logic   [9:0]  posX, posY,
550     input  logic   [2:0]  shape
551     );
552
553     /*************************************
554      *       Internal Signals
555      *************************************/
556
557     color_t shapeColor;
558     shape_t shapeType;
559     logic   inBoxX, inBoxY;
560     logic   inTopStick, inBottomStick, inLeftStick, inRightStick;
561
562     /*************************************
563      *       Boundary Checks
564      *************************************/
565
566     // create square boundaries
```

```
567     offset_check #(10) squareCheckX (
568             .val        (x),
569             .low        (posX),
570             .delta      (SIDE),
571             .is_between (inBoxX));
572 
573     offset_check #(10) squareCheckY (
574             .val        (y),
575             .low        (posY),
576             .delta      (SIDE),
577             .is_between (inBoxY));
578 
579     // create a top border inside the box
580     offset_check #(10) topStick (
581             .val        (y),
582             .low        (posY),
583             .delta      (LINEWIDTH),
584             .is_between (inTopStick));
585 
586     // create a bottom border inside the box
587     offset_check #(10) bottomStick (
588             .val        (y),
589             .low        (posY + SIDE - LINEWIDTH),
590             .delta      (LINEWIDTH),
591             .is_between (inBottomStick));
592 
593     // create a left border inside the box
594     offset_check #(10) leftStick (
595             .val        (x),
596             .low        (posX),
597             .delta      (LINEWIDTH),
598             .is_between (inLeftStick));
599 
600     // create a right border inside the box
601     offset_check #(10) rightStick (
602             .val        (x),
603             .low        (posX + SIDE - LINEWIDTH),
604             .delta      (LINEWIDTH),
605             .is_between (inRightStick));
606 
607     /*****************************************
608      *       Final Output logic
609      *****************************************/
610 
611     assign color = shapeColor;
612     assign shapeType = shape_t'(shape);
613 
614     always_comb begin
615         shapeColor = BLACK;
616 
617         if (inBoxX & inBoxY) begin
618             if ((shapeType == WALL) & (inLeftStick | inRightStick)) begin
619                 shapeColor = RED;
620             end
621             else if ((shapeType == LEFTTOP) & (inLeftStick | inTopStick)) begin
622                 shapeColor = BLUE;
623             end
624             else if ((shapeType == RIGHTTOP) & (inRightStick | inTopStick))
625             begin
626                 shapeColor = CYAN;
627             end
628             else if ((shapeType == RIGHTBOT) & (inRightStick | inBottomStick))
629             begin
630                 shapeColor = GREEN;
631             end
632             else if ((shapeType == LEFTBOT) & (inLeftStick | inBottomStick))
633             begin
634                 shapeColor = YELLOW;
635             end
636             else if ((shapeType == EQUAL) & (inBottomStick | inTopStick)) begin
637                 shapeColor = PURPLE;
```

```
            end
        end
    end
endmodule: drawShape

/*****************************************************************
 *
 *                     VGA Magic
 *
 *****************************************************************/

/** BRIEF
 *  VGA module that outputs the current hsync and vsync values needed
 *  to display content. Does not handle the actual color content.
 *
 *  Requires the Library.sv modules to work. Supports 640 x 480 px.
 */
module vga (
    output logic [9:0] row, col,
    output logic       HS, VS, blank,
    input  logic       CLOCK_50, reset
    );

    logic [10:0] col_count;
    logic        col_clear, col_enable;
    logic [9:0]  row_count;
    logic        row_clear, row_enable;
    logic        h_blank, v_blank;

    // Row counter counts from 0 to 520
    //     count of   0 - 479 is display time (thus row_count is correct here)
    //     count of 480 - 489 is front porch
    //     count of 490 - 491 is VS=0 pulse width
    //     count of 492 - 520 is back porch

    simple_counter #(10) row_counter(
            .Q      (row_count),
            .en     (row_enable),
            .clr    (row_clear),
            .clk    (CLOCK_50),
            .reset  (reset)
            );

    assign row        = row_count;
    assign row_clear  = (row_count >= 10'd520);
    assign row_enable = (col_count == 11'd1599);
    assign VS         = (row_count < 10'd490) | (row_count > 10'd491);
    assign v_blank    = (row_count >= 10'd480);

    // Col counter counts from 0 to 1599
    //     count of    0 - 1279 is display time (col is div by 2)
    //     count of 1280 - 1311 is front porch
    //     count of 1312 - 1503 is HS=0 pulse width
    //     count of 1504 - 1599 is back porch

    simple_counter #(11) col_counter(
            .Q      (col_count),
            .en     (col_enable),
            .clr    (col_clear),
            .clk    (CLOCK_50),
            .reset  (reset)
            );

    assign col        = col_count[10:1];
    assign col_clear  = (col_count >= 11'd1599);
    assign col_enable = 1'b1;
    assign HS         = (col_count < 11'd1312) | (col_count > 11'd1503);
    assign h_blank    = col_count > 11'd1279;

    assign blank      = h_blank | v_blank;
endmodule: vga
```

```
709
710 /////////////////////////////////////////////////////////////////
711  //
712  //                    Library modules
713  //
714  /////////////////////////////////////////////////////////////////
715
716 /** BRIEF
717  *  Outputs whether a value lies between [low, high].
718  */
719 module range_check
720     #(parameter WIDTH = 4'd10) (
721     input  logic [WIDTH-1:0] val, low, high,
722     output logic             is_between
723     );
724
725     assign is_between = (val >= low) & (val <= high);
726
727 endmodule: range_check
728
729 /** BRIEF
730  *  Outputs whether a value lies between [low, low + delta].
731  */
732 module offset_check
733     #(parameter WIDTH = 4'd10) (
734     input  logic [WIDTH-1:0] val, low, delta,
735     output logic             is_between
736     );
737
738     assign is_between = ((val >= low) & (val < (low+delta)));
739
740 endmodule: offset_check
741
742 /** BRIEF
743  *  Simple up counter with synchronous clear and enable.
744  *  Clear takes precedence over enable.
745  */
746 module simple_counter
747     #(parameter WIDTH = 4'd8) (
748     output logic [WIDTH-1:0] Q,
749     input  logic             clk, en, clr, reset
750     );
751
752     always_ff @(posedge clk, posedge reset)
753         if (reset)
754             Q <= 'b0;
755         else if (clr)
756             Q <= 'b0;
757         else if (en)
758             Q <= (Q + 1'b1);
759
760 endmodule: simple_counter
761
762 /** BRIEF
763  *  A register with synchronous clear. Clear takes precedence.
764  */
765 module registerAZ
766     #(parameter WIDTH = 4'd8) (
767     output logic [WIDTH-1:0] Q,
768     input  logic [WIDTH-1:0] D,
769     input  logic             clk, en, clr, reset
770     );
771
772     always_ff @(posedge clk, posedge reset) begin
773         if (reset)
774             Q <= 'b0;
775         else if (clr)
776             Q <= 'b0;
777         else if (en)
778             Q <= D;
779     end
```

```
780 endmodule: registerAZ
```

Lab Code [10 points]
Filename: zorgGame.sv
AndrewID: tbeasley

```
`default_nettype none

module ChipInterface // Chip Interface for FPGA and VGA connections
  (input logic [17:0] SW,
  input logic [3:0] KEY,
  input logic CLOCK_50,
  output logic [7:0] LEDG,
  output logic [8:0] LEDR,
  output logic [6:0] HEX3, HEX2, HEX1, HEX0,
  output logic [7:0]  VGA_R, VGA_G, VGA_B,
  output logic        VGA_BLANK_N, VGA_CLK, VGA_SYNC_N,
  output logic        VGA_VS, VGA_HS);

  logic [3:0] znarlyOut, zoodOut, roundOut, gamesOut;
  logic [11:0] masterOut;
  logic GameWonOut, maxRoundOut, clearGameOut, done, loadValue, syncGradeIt;

  assign clearGameOut = (GameWonOut|maxRoundOut);
  assign LEDG[0] = GameWonOut;


  ZorgGame game(.CoinValue(SW[17:16]), .ShapeLocation(SW[4:3]),
                .CoinInserted(KEY[1]), .StartGame(KEY[2]), .GradeIt(KEY[3]),
                .LoadShapeNow(KEY[3]), .reset(KEY[0]), .clock(CLOCK_50),
                .Guess(SW[11:0]), .LoadShape(SW[2:0]),.Znarly(znarlyOut),
                .Zood(zoodOut), .RoundNumber(roundOut), .NumGames(gamesOut),
                .GameWon(GameWonOut), .maxRound(maxRoundOut),
                .masterPattern(masterOut), .doneMasterPattern(done),
                .syncGradeIt);


  mastermindVGA vgaDisplay(.CLOCK_50, .VGA_R, .VGA_G, .VGA_B, .VGA_BLANK_N,
                           .VGA_CLK, .VGA_SYNC_N, .VGA_VS, .VGA_HS,
                           .numGames(gamesOut), .loadNumGames(1'b1),
                           .roundNumber(roundOut), .guess(SW[11:0]),
                           .loadGuess(done), .znarly(znarlyOut),
                           .zood(zoodOut), .loadZnarlyZood(syncGradeIt),
                           .clearGame(clearGameOut), .masterPattern(masterOut),
                           .displayMasterPattern(SW[15]), .reset(~KEY[0]));


  SevenSegmentDisplay seg(.BCD0(gamesOut), .BCD1(roundOut), .BCD2(zoodOut),
                          .BCD3(znarlyOut), .blank(8'b1111_0000), .HEX0, .HEX1,
                          .HEX2, .HEX3);

endmodule: ChipInterface


// Expanded version of the abstract FSM implemented in Task 2
module myAbstractFSMExpanded (
  output logic [3:0] credit,
  output logic drop,
  input logic [1:0] CoinValue,
  input logic CoinInserted_L, clock, reset_L);

  enum logic [4:0] {NOCREDIT, HOLDC00, HOLDT00, HOLDP00,
                    CRED1DROP0, HOLDC10, HOLDT10, HOLDP10,
                    CRED2DROP0, HOLDC20, HOLDT20, HOLDP20,
                    CRED3DROP0, HOLDC30, HOLDT30, HOLDP30,
                    CRED0DROP1, HOLDC01, HOLDT01, HOLDP01,
                    CRED1DROP1, HOLDC11, HOLDT11, HOLDP11,
                    CRED2DROP1, HOLDC21, HOLDT21, HOLDP21,
                    CRED3DROP1, HOLDC31, HOLDT31, HOLDP31} currState, nextState;

//Next State Logic
always_comb begin
  case (currState)
    NOCREDIT: begin
        if (CoinValue == 2'b01 & ~CoinInserted_L)
```

```
 70            nextState = HOLDC00;
 71          else if (CoinValue == 2'b10 & ~CoinInserted_L)
 72            nextState = HOLDT00;
 73          else if (CoinValue == 2'b11 & ~CoinInserted_L)
 74            nextState = HOLDP00;
 75          else
 76            nextState = NOCREDIT;
 77      end
 78      HOLDC00: begin
 79          if (~CoinInserted_L)
 80            nextState = HOLDC00;
 81          else
 82            nextState = CRED1DROP0;
 83      end
 84      HOLDT00: begin
 85          if (~CoinInserted_L)
 86            nextState = HOLDT00;
 87          else
 88            nextState = CRED3DROP0;
 89      end
 90      HOLDP00: begin
 91          if (~CoinInserted_L)
 92            nextState = HOLDP00;
 93          else
 94            nextState = CRED1DROP1;
 95      end
 96
 97      CRED1DROP0: begin
 98          if (CoinValue == 2'b01 & ~CoinInserted_L)
 99            nextState = HOLDC10;
100          else if (CoinValue == 2'b10 & ~CoinInserted_L)
101            nextState = HOLDT10;
102          else if (CoinValue == 2'b11 & ~CoinInserted_L)
103            nextState = HOLDP10;
104          else
105            nextState = CRED1DROP0;
106      end
107      HOLDC10: begin
108          if (~CoinInserted_L)
109            nextState = HOLDC10;
110          else
111            nextState = CRED2DROP0;
112      end
113      HOLDT10: begin
114          if (~CoinInserted_L)
115            nextState = HOLDT10;
116          else
117            nextState = CRED0DROP1;
118      end
119      HOLDP10: begin
120          if (~CoinInserted_L)
121            nextState = HOLDP10;
122          else
123            nextState = CRED2DROP1;
124      end
125
126      CRED2DROP0: begin
127          if (CoinValue == 2'b01 & ~CoinInserted_L)
128            nextState = HOLDC20;
129          else if (CoinValue == 2'b10 & ~CoinInserted_L)
130            nextState = HOLDT20;
131          else if (CoinValue == 2'b11 & ~CoinInserted_L)
132            nextState = HOLDP20;
133          else
134            nextState = CRED2DROP0;
135      end
136      HOLDC20: begin
137          if (~CoinInserted_L)
138            nextState = HOLDC20;
139          else
140            nextState = CRED3DROP0;
```

```
141     end
142     HOLDT20: begin
143         if (~CoinInserted_L)
144           nextState = HOLDT20;
145         else
146           nextState = CRED1DROP1;
147     end
148     HOLDP20: begin
149         if (~CoinInserted_L)
150           nextState = HOLDP20;
151         else
152           nextState = CRED3DROP1;
153     end
154
155     CRED3DROP0: begin
156         if (CoinValue == 2'b01 & ~CoinInserted_L)
157           nextState = HOLDC30;
158         else if (CoinValue == 2'b10 & ~CoinInserted_L)
159           nextState = HOLDT30;
160         else if (CoinValue == 2'b11 & ~CoinInserted_L)
161           nextState = HOLDP30;
162         else
163           nextState = CRED3DROP0;
164     end
165     HOLDC30: begin
166         if (~CoinInserted_L)
167           nextState = HOLDC30;
168         else
169           nextState = CRED0DROP1;
170     end
171     HOLDT30: begin
172         if (~CoinInserted_L)
173           nextState = HOLDT30;
174         else
175           nextState = CRED2DROP1;
176     end
177     HOLDP30: begin
178         if (~CoinInserted_L)
179           nextState = HOLDP30;
180         else
181           nextState = CRED0DROP1;
182     end
183
184     CRED0DROP1: begin
185         if (CoinValue == 2'b01 & ~CoinInserted_L)
186           nextState = HOLDC01;
187         else if (CoinValue == 2'b10 & ~CoinInserted_L)
188           nextState = HOLDT01;
189         else if (CoinValue == 2'b11 & ~CoinInserted_L)
190           nextState = HOLDP01;
191         else
192           nextState = NOCREDIT;
193     end
194     HOLDC01: begin
195         if (~CoinInserted_L)
196           nextState = HOLDC01;
197         else
198           nextState = CRED1DROP0;
199     end
200     HOLDT01: begin
201         if (~CoinInserted_L)
202           nextState = HOLDT01;
203         else
204           nextState = CRED3DROP0;
205     end
206     HOLDP01: begin
207         if (~CoinInserted_L)
208           nextState = HOLDP01;
209         else
210           nextState = CRED1DROP1;
211     end
```

```
212
213     CRED1DROP1: begin
214         if (CoinValue == 2'b01 & ~CoinInserted_L)
215           nextState = HOLDC11;
216         else if (CoinValue == 2'b10 & ~CoinInserted_L)
217           nextState = HOLDT11;
218         else if (CoinValue == 2'b11 & ~CoinInserted_L)
219           nextState = HOLDP11;
220         else
221           nextState = CRED1DROP0;
222     end
223     HOLDC11: begin
224         if (~CoinInserted_L)
225           nextState = HOLDC11;
226         else
227           nextState = CRED2DROP0;
228     end
229     HOLDT11: begin
230         if (~CoinInserted_L)
231           nextState = HOLDT11;
232         else
233           nextState = CRED0DROP1;
234     end
235     HOLDP11: begin
236         if (~CoinInserted_L)
237           nextState = HOLDP11;
238         else
239           nextState = CRED2DROP1;
240     end
241
242     CRED2DROP1: begin
243         if (CoinValue == 2'b01 & ~CoinInserted_L)
244           nextState = HOLDC21;
245         else if (CoinValue == 2'b10 & ~CoinInserted_L)
246           nextState = HOLDT21;
247         else if (CoinValue == 2'b11 & ~CoinInserted_L)
248           nextState = HOLDP21;
249         else
250           nextState = CRED2DROP0;
251     end
252     HOLDC21: begin
253         if (~CoinInserted_L)
254           nextState = HOLDC21;
255         else
256           nextState = CRED3DROP0;
257     end
258     HOLDT21: begin
259         if (~CoinInserted_L)
260           nextState = HOLDT21;
261         else
262           nextState = CRED1DROP1;
263     end
264     HOLDP21: begin
265         if (~CoinInserted_L)
266           nextState = HOLDP21;
267         else
268           nextState = CRED3DROP1;
269     end
270
271     CRED3DROP1: begin
272         if (CoinValue == 2'b01 & ~CoinInserted_L)
273           nextState = HOLDC31;
274         else if (CoinValue == 2'b10 & ~CoinInserted_L)
275           nextState = HOLDT31;
276         else if (CoinValue == 2'b11 & ~CoinInserted_L)
277           nextState = HOLDP31;
278         else
279           nextState = CRED3DROP0;
280     end
281     HOLDC31: begin
282         if (~CoinInserted_L)
```

```
283           nextState = HOLDC31;
284         else
285           nextState = CRED0DROP1;
286     end
287     HOLDT31: begin
288         if (~CoinInserted_L)
289           nextState = HOLDT31;
290         else
291           nextState = CRED2DROP1;
292     end
293     HOLDP31: begin
294         if (~CoinInserted_L)
295           nextState = HOLDP31;
296         else
297           nextState = CRED0DROP1;
298     end
299
300     default: begin
301         nextState = NOCREDIT;
302     end
303   endcase
304 end
305
306 //Output logic
307 always_comb begin
308   credit = 4'b0000; drop = 1'b0;
309   unique case (currState)
310     NOCREDIT: begin
311       drop = 1'b0;
312       credit = 4'b0000;
313     end
314     HOLDC00: begin
315       drop = 1'b0;
316       credit = 4'b0000;
317     end
318     HOLDT00: begin
319       drop = 1'b0;
320       credit = 4'b0000;
321     end
322     HOLDP00: begin
323       drop = 1'b0;
324       credit = 4'b0000;
325     end
326
327     CRED1DROP0: begin
328       drop = 1'b0;
329       credit = 4'b0001;
330     end
331     HOLDC10: begin
332       drop = 1'b0;
333       credit = 4'b0001;
334     end
335     HOLDT10: begin
336       drop = 1'b0;
337       credit = 4'b0001;
338     end
339     HOLDP10: begin
340       drop = 1'b0;
341       credit = 4'b0001;
342     end
343
344     CRED2DROP0: begin
345       drop = 1'b0;
346       credit = 4'b0010;
347     end
348     HOLDC20: begin
349       drop = 1'b0;
350       credit = 4'b0010;
351     end
352     HOLDT20: begin
353       drop = 1'b0;
```

```
354       credit = 4'b0010;
355     end
356     HOLDP20: begin
357       drop = 1'b0;
358       credit = 4'b0010;
359     end
360
361     CRED3DROP0: begin
362       drop = 1'b0;
363       credit = 4'b0011;
364     end
365     HOLDC30: begin
366       drop = 1'b0;
367       credit = 4'b0011;
368     end
369     HOLDT30: begin
370       drop = 1'b0;
371       credit = 4'b0011;
372     end
373     HOLDP30: begin
374       drop = 1'b0;
375       credit = 4'b0011;
376     end
377
378     CRED0DROP1: begin
379       drop = 1'b1;
380       credit = 4'b0000;
381     end
382     HOLDC01: begin
383       drop = 1'b0;
384       credit = 4'b0000;
385     end
386     HOLDT01: begin
387       drop = 1'b0;
388       credit = 4'b0000;
389     end
390     HOLDP01: begin
391       drop = 1'b0;
392       credit = 4'b0000;
393     end
394
395     CRED1DROP1: begin
396       drop = 1'b1;
397       credit = 4'b0001;
398     end
399     HOLDC11: begin
400       drop = 1'b0;
401       credit = 4'b0001;
402     end
403     HOLDT11: begin
404       drop = 1'b0;
405       credit = 4'b0001;
406     end
407     HOLDP11: begin
408       drop = 1'b0;
409       credit = 4'b0001;
410     end
411
412     CRED2DROP1: begin
413       drop = 1'b1;
414       credit = 4'b0010;
415     end
416     HOLDC21: begin
417       drop = 1'b0;
418       credit = 4'b0010;
419     end
420     HOLDT21: begin
421       drop = 1'b0;
422       credit = 4'b0010;
423     end
424     HOLDP21: begin
```

```
425       drop = 1'b0;
426       credit = 4'b0010;
427     end
428
429     CRED3DROP1: begin
430       drop = 1'b1;
431       credit = 4'b0011;
432     end
433     HOLDC31: begin
434       drop = 1'b0;
435       credit = 4'b0011;
436     end
437     HOLDT31: begin
438       drop = 1'b0;
439       credit = 4'b0011;
440     end
441     HOLDP31: begin
442       drop = 1'b0;
443       credit = 4'b0011;
444     end
445   endcase
446 end
447
448 always_ff @(posedge clock, negedge reset_L)
449     if (~reset_L)
450       currState <= NOCREDIT;
451     else
452       currState <= nextState;
453
454 endmodule: myAbstractFSMExpanded
455
456
457 module ZorgGame // Task 2 Datapath
458   (input logic [1:0] CoinValue, ShapeLocation,
459   input logic CoinInserted, StartGame, GradeIt, LoadShapeNow, reset, clock,
460   input logic [11:0] Guess,
461   input logic [2:0] LoadShape,
462   output logic [3:0] Znarly, Zood, RoundNumber, NumGames,
463   output logic [11:0] masterPattern,
464   output logic GameWon, maxRound, doneMasterPattern, syncGradeIt);
465
466   logic numGameEn, numGameClr, numGameUp, notPaid, maxGames, shapeEn1,
467         shapeEn2, shapeClr1, shapeClr2, firstLoc,
468         secondLoc, thirdLoc, fourthLoc, validLocation, roundEn, roundClr,
469         guessedIt, underRoundLimit, syncCoinInserted,
470         syncStartGame, syncLoadShapeNow, syncReset,
471         drop, nextGame, Gclr, Gload, ldZnarly, ldZood, clrZnarly, clrZood;
472
473   logic [11:0] newShape, moveShapeOut, savedShiftedShapeOut,
474                choosePositionOut;
475   logic [1:0] checkLocationOut;
476   logic [3:0] shiftedByValue, numGamesOut, credit;
477
478   //Lab3 (Input here)
479   myAbstractFSMExpanded takeCoins(.credit, .drop, .CoinValue,
480                                   .CoinInserted_L(syncCoinInserted),
481                                   .clock, .reset_L(syncReset));
482
483   Synchronizer sync1(.async(CoinInserted), .clock, .sync(syncCoinInserted));
484   Synchronizer sync2(.async(StartGame), .clock, .sync(syncStartGame));
485   Synchronizer sync3(.async(GradeIt), .clock, .sync(syncGradeIt));
486   Synchronizer sync4(.async(LoadShapeNow), .clock, .sync(syncLoadShapeNow));
487   Synchronizer sync5(.async(reset), .clock, .sync(syncReset));
488
489   masterPatternFSM f1(.StartGame_L(syncStartGame), .GradeIt_L(syncGradeIt),
490                   .LoadShapeNow_L(syncLoadShapeNow), .reset_L(syncReset),
491                   .GameWon, .*);
492   numOfGameFSM f2(.reset_L(syncReset), .*);
493
494   Counter #(4) numberOfGames(.en(numGameEn), .clear(numGameClr),
495                              .up(numGameUp), .load(1'b0), .clock, .D(),
```

```
496                               .Q(numGamesOut));
497
498   assign NumGames = numGamesOut;
499
500   Counter #(4) numberOfRounds(.en(roundEn), .clear(roundClr), .load(1'b0),
501                               .up(1'b1), .clock, .D(), .Q(RoundNumber));
502
503   Comparator #(4) paidOrNot(.A(numGamesOut), .B(4'd0), .AeqB(notPaid));
504   Comparator #(4) maxNumberOfGames(.A(numGamesOut), .B(4'd7), .AeqB(maxGames));
505   Comparator #(4) guess(.A(Znarly), .B(4'd4), .AeqB(guessedIt));
506
507   Comparator #(3) locationOne(.A(masterPattern[2:0]), .B(3'd0),
508                               .AeqB(firstLoc));
509   Comparator #(3) locationTwo(.A(masterPattern[5:3]), .B(3'd0),
510                               .AeqB(secondLoc));
511   Comparator #(3) locationThird(.A(masterPattern[8:6]), .B(3'd0),
512                               .AeqB(thirdLoc));
513   Comparator #(3) locationFourth(.A(masterPattern[11:9]), .B(3'd0),
514                               .AeqB(fourthLoc));
515
516   MagComp #(4) c1(.A(RoundNumber), .B(4'd8), .AltB(underRoundLimit), .AgtB(),
517                .AeqB(maxRound));
518
519   always_comb begin
520     newShape[11:3] = 9'b0;
521     newShape[2:0] = LoadShape[2:0];
522     shiftedByValue = 4'd0;
523     if (ShapeLocation == 2'b00)
524       shiftedByValue = 4'd0;
525     else if (ShapeLocation == 2'b01)
526       shiftedByValue = 4'd3;
527     else if (ShapeLocation == 2'b10)
528       shiftedByValue = 4'd6;
529     else if (ShapeLocation == 2'b11)
530       shiftedByValue = 4'd9;
531   end
532
533   BarrelShifter moveShape(.V(newShape), .by(shiftedByValue),
534                           .S(moveShapeOut));
535
536   Register #(12) saveShiftValue(.en(shapeEn1), .clear(shapeClr1), .clock,
537                                 .D(moveShapeOut), .Q(savedShiftedShapeOut));
538   Register #(12) finalPos(.en(shapeEn2), .clear(shapeClr2), .clock,
539                           .D(choosePositionOut), .Q(masterPattern));
540
541   assign choosePositionOut = masterPattern | savedShiftedShapeOut;
542
543   Multiplexer #(4) m1(.I({fourthLoc, thirdLoc, secondLoc, firstLoc}),
544                       .S(ShapeLocation), .Y(validLocation));
545
546   assign doneMasterPattern = ~(firstLoc | secondLoc | thirdLoc | fourthLoc);
547   assign nextGame = GameWon | maxRound;
548
549   Grader grade(.Guess, .masterPattern, .Gclr, .Gload,
550                .clock(clock), .Znarly(Znarly), .Zood(Zood), .doneMasterPattern,
551                .ldZnarly, .ldZood, .clrZnarly, .clrZood);
552
553 endmodule: ZorgGame
554
555
556 module masterPatternFSM // FSM for controlling game based on masterPattern input
557   (input logic StartGame_L, GradeIt_L, LoadShapeNow_L, notPaid,
558    doneMasterPattern, validLocation, guessedIt, underRoundLimit, maxRound,
559    reset_L, clock,
560    output logic GameWon, shapeClr1, shapeClr2, roundEn, roundClr, shapeEn1,
561    shapeEn2, Gclr, Gload, ldZnarly, ldZood, clrZnarly, clrZood);
562
563   enum logic [3:0] {INIT, PREP, FINISHPREP, FINISH, EXIT, INCROUND, WON, HOLD1,
564                     HOLD2} currState, nextState;
565
566   //Next State Logic
```

```
567   always_comb begin
568     nextState = INIT;
569     case(currState)
570       INIT: begin
571         if (StartGame_L)
572           nextState = INIT;
573         else if (~StartGame_L && ~notPaid)
574           nextState = PREP;
575       end
576       PREP: begin
577         if (LoadShapeNow_L)
578           nextState = PREP;
579         else if (~LoadShapeNow_L)
580           nextState = FINISHPREP;
581       end
582       FINISHPREP: begin
583         if (~validLocation & LoadShapeNow_L)
584           nextState = PREP;
585         else if (validLocation & LoadShapeNow_L)
586           nextState = FINISH;
587         else if (~LoadShapeNow_L)
588           nextState = FINISHPREP;
589       end
590       FINISH: begin
591         if (~doneMasterPattern)
592           nextState = HOLD1;
593         else if (doneMasterPattern)
594           nextState = EXIT;
595       end
596       HOLD1: begin
597         if (~doneMasterPattern)
598           nextState = PREP;
599         else if (doneMasterPattern)
600           nextState = EXIT;
601       end
602       EXIT: begin
603         if (maxRound)
604           nextState = INIT;
605         else if (~GradeIt_L & ~guessedIt & ~maxRound)
606           nextState = HOLD2;
607         else if (GradeIt_L)
608           nextState = EXIT;
609       end
610       HOLD2: begin
611         if (~GradeIt_L)
612           nextState = HOLD2;
613         else if(GradeIt_L)
614           nextState = INCROUND;
615       end
616       INCROUND:
617         if (guessedIt & underRoundLimit)
618           nextState = WON;
619         else
620           nextState = EXIT;
621       WON: begin
622         nextState = INIT;
623       end
624     endcase
625   end
626
627   //Output Logic
628   always_comb begin
629     shapeClr1 = 0; shapeClr2 = 0; roundClr = 0; roundEn = 0;
630     shapeEn1 = 0; shapeEn2 = 0; GameWon = 0; Gclr = 0; Gload = 0; ldZnarly = 0;
631     ldZood = 0; clrZnarly = 0; clrZood = 0;
632     unique case(currState)
633       INIT: begin
634         shapeClr1 = 1;
635         shapeClr2 = 1;
636         roundClr = 1;
637         roundEn = 0;
```

```
638         shapeEn1 = 0;
639         shapeEn2 = 0;
640         GameWon = 0;
641         Gclr = 1;
642         Gload = 0;
643         ldZnarly = 0;
644         ldZood = 0;
645         clrZnarly = 1;
646         clrZood = 1;
647       end
648       PREP: begin
649         shapeClr1 = 1;
650         shapeClr2 = 0;
651         roundClr = 0;
652         roundEn = 0;
653         shapeEn1 = 0;
654         shapeEn2 = 0;
655         Gclr = 0;
656         Gload = 0;
657         ldZnarly = 0;
658         ldZood = 0;
659         clrZnarly = 0;
660         clrZood = 0;
661       end
662       FINISHPREP: begin
663         shapeClr1 = 0;
664         shapeClr2 = 0;
665         roundClr = 0;
666         roundEn = 0;
667         shapeEn1 = 1;
668         shapeEn2 = 0;
669         Gclr = 0;
670         Gload = 0;
671         ldZnarly = 0;
672         ldZood = 0;
673         clrZnarly = 0;
674         clrZood = 0;
675       end
676       FINISH: begin
677         shapeClr1 = 1;
678         shapeClr2 = 0;
679         roundClr = 0;
680         roundEn = 0;
681         shapeEn1 = 0;
682         shapeEn2 = 1;
683         Gclr = 0;
684         Gload = 0;
685         ldZnarly = 0;
686         ldZood = 0;
687         clrZnarly = 0;
688         clrZood = 0;
689       end
690       HOLD1: begin
691         shapeClr1 = 1;
692         shapeClr2 = 0;
693         roundClr = 0;
694         roundEn = 0;
695         shapeEn1 = 0;
696         shapeEn2 = 1;
697         Gclr = 0;
698         Gload = 0;
699         ldZnarly = 0;
700         ldZood = 0;
701         clrZnarly = 0;
702         clrZood = 0;
703       end
704       EXIT: begin
705         shapeClr1 = 0;
706         shapeClr2 = 0;
707         roundClr = 0;
708         roundEn = 0;
```

```
709         shapeEn1 = 0;
710         shapeEn2 = 0;
711         Gclr = 1;
712         Gload = 0;
713         ldZnarly = 0;
714         ldZood = 0;
715         clrZnarly = 0;
716         clrZood = 0;
717       end
718       HOLD2: begin
719         shapeClr1 = 0;
720         shapeClr2 = 0;
721         roundClr = 0;
722         roundEn = 0;
723         shapeEn1 = 0;
724         shapeEn2 = 0;
725         Gclr = 0;
726         Gload = 1;
727         ldZnarly = 1;
728         ldZood = 1;
729         clrZnarly = 0;
730         clrZood = 0;
731       end
732       INCROUND: begin
733         shapeClr1 = 0;
734         shapeClr2 = 0;
735         roundClr = 0;
736         roundEn = 1;
737         shapeEn1 = 0;
738         shapeEn2 = 0;
739         Gclr = 1;
740         Gload = 0;
741         ldZnarly = 0;
742         ldZood = 0;
743         clrZnarly = 1;
744         clrZood = 1;
745       end
746       WON: begin
747         shapeClr1 = 1;
748         shapeClr2 = 1;
749         roundClr = 0;
750         roundEn = 0;
751         shapeEn1 = 0;
752         shapeEn2 = 0;
753         GameWon = 1;
754         Gclr = 1;
755         Gload = 0;
756         ldZnarly = 0;
757         ldZood = 0;
758         clrZnarly = 1;
759         clrZood = 1;
760       end
761     endcase
762   end
763
764   always_ff @(posedge clock, negedge reset_L)
765     if (~reset_L)
766       currState <= INIT;
767     else
768       currState <= nextState;
769
770 endmodule: masterPatternFSM
771
772
773 module numOfGameFSM // FSM for controlling number of games
774   (input logic drop, maxGames, nextGame, reset_L, clock,
775   output logic numGameEn, numGameClr, numGameUp);
776
777   enum logic [1:0] {INIT, PAID, STOP, REMOVEGAME} currState, nextState;
778
779   //Next State Logic
```

```
780   always_comb begin
781     nextState = INIT;
782     case(currState)
783       INIT: begin
784         if (~drop | maxGames)
785           nextState = INIT;
786         else if (drop & ~maxGames)
787           nextState = PAID;
788       end
789       PAID: begin
790         if (drop & ~maxGames & ~nextGame)
791           nextState = PAID;
792         else if (~nextGame & (~drop | maxGames))
793           nextState = STOP;
794         else if (nextGame)
795           nextState = REMOVEGAME;
796       end
797       STOP: begin
798         if (~nextGame & (~drop | maxGames))
799           nextState = STOP;
800         else if (drop & ~maxGames & ~nextGame)
801           nextState = PAID;
802         else if (nextGame)
803           nextState = REMOVEGAME;
804       end
805       REMOVEGAME: begin
806         nextState = STOP;
807       end
808     endcase
809   end
810
811   //Output Logic
812   always_comb begin
813     numGameEn = 0; numGameClr = 0; numGameUp = 0;
814     unique case(currState)
815       INIT: begin
816         numGameEn = 0;
817         numGameClr = 1;
818         numGameUp = 0;
819       end
820       PAID: begin
821         numGameEn = 1;
822         numGameClr = 0;
823         numGameUp = 1;
824       end
825       STOP: begin
826         numGameEn = 0;
827         numGameClr = 0;
828         numGameUp = 1;
829       end
830       REMOVEGAME: begin
831         numGameEn = 1;
832         numGameClr = 0;
833         numGameUp = 0;
834       end
835     endcase
836   end
837
838   always_ff @(posedge clock, negedge reset_L)
839     if (~reset_L)
840       currState <= INIT;
841     else
842       currState <= nextState;
843
844 endmodule: numOfGameFSM
845
846
847 // Count Znarlys and Zoods in Guess compared to masterPattern
848 module Grader
849   (input logic [11:0] Guess, masterPattern,
850    input logic Gclr, Gload, clock, doneMasterPattern, ldZnarly, ldZood,
```

```
851     input logic clrZnarly, clrZood,
852     output logic AeqB1, AeqB2, AeqB3, AeqB4,
853     output logic [11:0] Guess_pos,
854     output logic [3:0] Znarly, Zood, sum1, sum2, sum3, sum4, sum5, sum6, sum7);
855
856     logic [3:0] znarlyOut, zoodOut, finalZnarly, finalZood;
857
858     // Znarly Counter //
859     Register #(12) r1(.D(Guess), .Q(Guess_pos),
860                       .clock(clock), .en(Gload), .clear(Gclr));
861
862     Comparator #(4) znc1(.A(Guess_pos[2:0]), .B(masterPattern[2:0]),
863                          .AeqB(AeqB1));
864     Comparator #(4) znc2(.A(Guess_pos[5:3]), .B(masterPattern[5:3]),
865                          .AeqB(AeqB2));
866     Comparator #(4) znc3(.A(Guess_pos[8:6]), .B(masterPattern[8:6]),
867                          .AeqB(AeqB3));
868     Comparator #(4) znc4(.A(Guess_pos[11:9]), .B(masterPattern[11:9]),
869                          .AeqB(AeqB4));
870
871     Adder #(4) zna1(.A(AeqB1), .B(AeqB2), .sum(sum1), .cin(0), .cout());
872     Adder #(4) zna2(.A(AeqB3), .B(AeqB4), .sum(sum2), .cin(0), .cout());
873     Adder #(4) zna3(.A(sum1), .B(sum2), .sum(znarlyOut), .cin(0), .cout());
874
875     // Zood Counter //
876     logic [2:0] Tshape, Cshape, Oshape, Dshape, Ishape, Zshape;
877     logic [3:0] T_count, C_count, O_count, D_count, I_count, Z_count;
878
879     assign Tshape = 3'b001;
880     assign Cshape = 3'b010;
881     assign Oshape = 3'b011;
882     assign Dshape = 3'b100;
883     assign Ishape = 3'b101;
884     assign Zshape = 3'b110;
885
886     // Compare number of specific shape in Guess vs. masterPattern
887     Shape_Counter tc(.Shape(Tshape),
888                      .partial_Zoods(T_count), .Guess_pos(Guess_pos),
889                      .masterPattern(masterPattern));
890     Shape_Counter cc(.Shape(Cshape),
891                      .partial_Zoods(C_count), .Guess_pos(Guess_pos),
892                      .masterPattern(masterPattern));
893     Shape_Counter oc(.Shape(Oshape),
894                      .partial_Zoods(O_count), .Guess_pos(Guess_pos),
895                      .masterPattern(masterPattern));
896     Shape_Counter dc(.Shape(Dshape),
897                      .partial_Zoods(D_count), .Guess_pos(Guess_pos),
898                      .masterPattern(masterPattern));
899     Shape_Counter ic(.Shape(Ishape),
900                      .partial_Zoods(I_count), .Guess_pos(Guess_pos),
901                      .masterPattern(masterPattern));
902     Shape_Counter zc(.Shape(Zshape),
903                      .partial_Zoods(Z_count), .Guess_pos(Guess_pos),
904                      .masterPattern(masterPattern));
905
906     Adder #(3) zoa1(.A(T_count), .B(C_count), .sum(sum3), .cin(0), .cout());
907     Adder #(3) zoa2(.A(O_count), .B(D_count), .sum(sum4), .cin(0), .cout());
908     Adder #(3) zoa3(.A(I_count), .B(Z_count), .sum(sum5), .cin(0), .cout());
909     Adder #(3) zoa4(.A(sum3), .B(sum4), .sum(sum6), .cin(0), .cout());
910     Adder #(3) zoa5(.A(sum6), .B(sum5), .sum(sum7), .cin(0), .cout());
911
912     // Subtract Znarly count from shape count to obtain Zood count
913     Subtracter #(3) sub(.A(sum7), .B(znarlyOut), .diff(zoodOut), .bin(0),
914                         .bout());
915
916     Register #(4) z1(.D(znarlyOut), .Q(finalZnarly),
917                      .clock(clock), .en(ldZnarly), .clear(clrZnarly));
918     Register #(4) z2(.D(zoodOut), .Q(finalZood),
919                      .clock(clock), .en(ldZood), .clear(clrZood));
920
921     always_comb begin
```

```
922     if (~doneMasterPattern) begin
923       Znarly = 4'b0;
924       Zood = 4'b0;
925     end
926     else begin
927       Znarly = finalZnarly;
928       Zood = finalZood;
929     end
930    end
931
932 endmodule: Grader
933
934
935 // Count number of shapes in Guess and masterPattern
936 module Shape_Counter
937   (input logic [11:0] Guess_pos, masterPattern,
938    input logic [2:0] Shape,
939    output logic [2:0] partial_Zoods,
940    output logic mp_AeqB1, mp_AeqB2, mp_AeqB3, mp_AeqB4,
941    output logic [3:0] mp_sum1, mp_sum2, mp_sum3,
942    output logic g_AeqB1, g_AeqB2, g_AeqB3, g_AeqB4,
943    output logic [3:0] g_sum1, g_sum2, g_sum3,
944    output logic mag_AgtB);
945
946   // Count Shape in masterPattern
947   Comparator #(3) zocmp1(.A(Shape), .B(masterPattern[2:0]),
948                          .AeqB(mp_AeqB1));
949   Comparator #(3) zocmp2(.A(Shape), .B(masterPattern[5:3]),
950                          .AeqB(mp_AeqB2));
951   Comparator #(3) zocmp3(.A(Shape), .B(masterPattern[8:6]),
952                          .AeqB(mp_AeqB3));
953   Comparator #(3) zocmp4(.A(Shape), .B(masterPattern[11:9]),
954                          .AeqB(mp_AeqB4));
955
956   Adder #(3) zoamp1(.A(mp_AeqB1), .B(mp_AeqB2), .sum(mp_sum1),
957                     .cin(0), .cout());
958   Adder #(3) zoamp2(.A(mp_AeqB3), .B(mp_AeqB4), .sum(mp_sum2),
959                     .cin(0), .cout());
960   Adder #(3) zoamp3(.A(mp_sum1), .B(mp_sum2), .sum(mp_sum3),
961                     .cin(0), .cout());
962
963   // Count Shape in Guess
964   Comparator #(3) zocg1(.A(Shape), .B(Guess_pos[2:0]),
965                         .AeqB(g_AeqB1));
966   Comparator #(3) zocg2(.A(Shape), .B(Guess_pos[5:3]),
967                         .AeqB(g_AeqB2));
968   Comparator #(3) zocg3(.A(Shape), .B(Guess_pos[8:6]),
969                         .AeqB(g_AeqB3));
970   Comparator #(3) zocg4(.A(Shape), .B(Guess_pos[11:9]),
971                         .AeqB(g_AeqB4));
972
973   Adder #(3) zoag1(.A(g_AeqB1), .B(g_AeqB2), .sum(g_sum1),
974                    .cin(0), .cout());
975   Adder #(3) zoag2(.A(g_AeqB3), .B(g_AeqB4), .sum(g_sum2),
976                    .cin(0), .cout());
977   Adder #(3) zoag3(.A(g_sum1), .B(g_sum2), .sum(g_sum3),
978                    .cin(0), .cout());
979
980   // Compare masterPattern and Guess Shape counts
981   MagComp #(3) mg(.A(mp_sum3), .B(g_sum3),
982                   .AltB(), .AeqB(), .AgtB(mag_AgtB));
983
984   // Select lowest count
985   Mux2to1 #(3) mult(.I0(mp_sum3), .I1(g_sum3), .S(mag_AgtB), .Y(partial_Zoods));
986
987 endmodule: Shape_Counter
988
989
990 //Helps to display the variables we defined onto the FPGA board BCDs
991 module SevenSegmentDisplay
992 (input logic [3:0] BCD7, BCD6, BCD5, BCD4, BCD3, BCD2, BCD1, BCD0,
```

```
993 input logic [7:0] blank,
994 output logic [6:0] HEX7, HEX6, HEX5, HEX4, HEX3, HEX2, HEX1, HEX0);
995
996 logic [6:0] preHEX7, preHEX6, preHEX5, preHEX4, preHEX3, preHEX2, preHEX1,
997              preHEX0;
998 logic [6:0] nonInvertedHEX7,nonInvertedHEX6,nonInvertedHEX5,nonInvertedHEX4,
999              nonInvertedHEX3,nonInvertedHEX2, nonInvertedHEX1, nonInvertedHEX0;
1000
1001 BCDtoSevenSegment d0(.bcd(BCD0), .segment(preHEX0));
1002 BCDtoSevenSegment d1(.bcd(BCD1), .segment(preHEX1));
1003 BCDtoSevenSegment d2(.bcd(BCD2), .segment(preHEX2));
1004 BCDtoSevenSegment d3(.bcd(BCD3), .segment(preHEX3));
1005 BCDtoSevenSegment d4(.bcd(BCD4), .segment(preHEX4));
1006 BCDtoSevenSegment d5(.bcd(BCD5), .segment(preHEX5));
1007 BCDtoSevenSegment d6(.bcd(BCD6), .segment(preHEX6));
1008 BCDtoSevenSegment d7(.bcd(BCD7), .segment(preHEX7));
1009
1010 Mux2to1 m0(.I0(preHEX0), .I1(7'b0), .S(blank[0]), .Y(nonInvertedHEX0));
1011 Mux2to1 m1(.I0(preHEX1), .I1(7'b0), .S(blank[1]), .Y(nonInvertedHEX1));
1012 Mux2to1 m2(.I0(preHEX2), .I1(7'b0), .S(blank[2]), .Y(nonInvertedHEX2));
1013 Mux2to1 m3(.I0(preHEX3), .I1(7'b0), .S(blank[3]), .Y(nonInvertedHEX3));
1014 Mux2to1 m4(.I0(preHEX4), .I1(7'b0), .S(blank[4]), .Y(nonInvertedHEX4));
1015 Mux2to1 m5(.I0(preHEX5), .I1(7'b0), .S(blank[5]), .Y(nonInvertedHEX5));
1016 Mux2to1 m6(.I0(preHEX6), .I1(7'b0), .S(blank[6]), .Y(nonInvertedHEX6));
1017 Mux2to1 m7(.I0(preHEX7), .I1(7'b0), .S(blank[7]), .Y(nonInvertedHEX7));
1018
1019 assign HEX0 = ~nonInvertedHEX0;
1020 assign HEX1 = ~nonInvertedHEX1;
1021 assign HEX2 = ~nonInvertedHEX2;
1022 assign HEX3 = ~nonInvertedHEX3;
1023 assign HEX4 = ~nonInvertedHEX4;
1024 assign HEX5 = ~nonInvertedHEX5;
1025 assign HEX6 = ~nonInvertedHEX6;
1026 assign HEX7 = ~nonInvertedHEX7;
1027
1028 endmodule: SevenSegmentDisplay
1029
1030
1031 //Converts the BCDs into the seven segments for the displays on the FPGA
1032 module BCDtoSevenSegment
1033   (input logic [3:0] bcd,
1034   output logic [6:0] segment);
1035
1036   always_comb begin
1037     unique case(bcd)
1038       4'b0000: segment = 7'b011_1111;
1039       4'b0001: segment = 7'b000_0110;
1040       4'b0010: segment = 7'b101_1011;
1041       4'b0011: segment = 7'b100_1111;
1042       4'b0100: segment = 7'b110_0110;
1043       4'b0101: segment = 7'b110_1101;
1044       4'b0110: segment = 7'b111_1101;
1045       4'b0111: segment = 7'b000_0111;
1046       4'b1000: segment = 7'b111_1111;
1047       4'b1001: segment = 7'b110_0111;
1048       default: segment = 7'b000_0000;
1049     endcase
1050   end
1051 endmodule: BCDtoSevenSegment
1052
1053
1054 module ZorgGame_test; // Testbench for ZorgGame
1055   logic [1:0] CoinValue, ShapeLocation;
1056   logic CoinInserted, StartGame, GradeIt, LoadShapeNow, reset, clock, GameWon;
1057   logic maxRound;
1058   logic [11:0] Guess, masterPattern;
1059   logic [2:0] LoadShape;
1060   logic [3:0] Znarly, Zood, RoundNumber, NumGames;
1061   logic doneMasterPattern;
1062
1063   ZorgGame DUT(.*);
```

```
1064
1065   initial begin
1066     clock = 0;
1067     forever #5 clock = ~clock;
1068   end
1069
1070   initial begin
1071     $monitor($time,, "Coin = %b location = %b coinInserted = %b start = %b",
1072     CoinValue, ShapeLocation, CoinInserted, StartGame,
1073     " grade = %b loadShape = %b loadNow = %b guess = %b", GradeIt, LoadShape,
1074     LoadShapeNow, Guess,
1075     " Znarly = %d Zood = %d, RoundNumber = %d, NumGames = %d Won = %d",
1076     Znarly, Zood, RoundNumber, NumGames, GameWon);
1077
1078     //Initial Values
1079     reset <= 0; CoinValue <= 2'b01; ShapeLocation <= 2'b00; CoinInserted <= 1;
1080     StartGame <= 1; GradeIt <= 1; LoadShapeNow <= 1; LoadShape <= 3'b110;
1081     Guess <= 12'b010_001_110_001;
1082
1083     @(posedge clock);
1084
1085     reset <= 1;
1086     @(posedge clock);
1087     @(posedge clock);
1088     @(posedge clock);
1089     StartGame <= 0;
1090     @(posedge clock);
1091     @(posedge clock);
1092     @(posedge clock);
1093     @(posedge clock);
1094     LoadShapeNow <= 0;
1095     @(posedge clock);
1096     @(posedge clock);
1097     @(posedge clock);
1098     StartGame <= 1; LoadShapeNow <= 1; CoinInserted <= 0;
1099     @(posedge clock);
1100     @(posedge clock);
1101     @(posedge clock);
1102     @(posedge clock);
1103     CoinInserted <= 1;
1104     @(posedge clock);
1105     @(posedge clock);
1106     @(posedge clock);
1107     @(posedge clock);
1108     @(posedge clock);
1109     CoinValue <= 2'b10; CoinInserted <= 0;
1110     @(posedge clock);
1111     @(posedge clock);
1112     @(posedge clock);
1113     CoinInserted <= 1;
1114     @(posedge clock);
1115     @(posedge clock);
1116     @(posedge clock);
1117     @(posedge clock);
1118     @(posedge clock);
1119     @(posedge clock);
1120     @(posedge clock);
1121     CoinValue <= 2'b11; CoinInserted <= 0;
1122     @(posedge clock);
1123     @(posedge clock);
1124     CoinInserted <= 1;
1125     @(posedge clock);
1126     @(posedge clock);
1127     @(posedge clock);
1128     @(posedge clock);
1129     @(posedge clock);
1130     @(posedge clock);
1131     @(posedge clock);
1132     @(posedge clock);
1133     CoinInserted <= 0;
1134     @(posedge clock);
```

```
1135     @(posedge clock);
1136     @(posedge clock);
1137     CoinInserted <= 1;
1138     @(posedge clock);
1139     @(posedge clock);
1140     @(posedge clock);
1141     @(posedge clock);
1142     @(posedge clock);
1143     @(posedge clock);
1144     @(posedge clock);
1145     @(posedge clock);
1146     CoinInserted <= 0;
1147     @(posedge clock);
1148     @(posedge clock);
1149     @(posedge clock);
1150     CoinInserted <= 1;
1151     @(posedge clock);
1152     @(posedge clock);
1153     @(posedge clock);
1154     @(posedge clock);
1155     @(posedge clock);
1156     @(posedge clock);
1157     @(posedge clock);
1158     @(posedge clock);
1159     CoinInserted <= 0;
1160     @(posedge clock);
1161     @(posedge clock);
1162     @(posedge clock);
1163     CoinInserted <= 1;
1164     @(posedge clock);
1165     @(posedge clock);
1166     @(posedge clock);
1167     @(posedge clock);
1168     @(posedge clock);
1169     @(posedge clock);
1170     @(posedge clock);
1171     @(posedge clock);
1172     CoinInserted <= 0;
1173     @(posedge clock);
1174     @(posedge clock);
1175     @(posedge clock);
1176     CoinInserted <= 1;
1177     @(posedge clock);
1178     @(posedge clock);
1179     @(posedge clock);
1180     @(posedge clock);
1181     @(posedge clock);
1182     @(posedge clock);
1183     @(posedge clock);
1184     @(posedge clock);
1185     CoinInserted <= 0;
1186     @(posedge clock);
1187     @(posedge clock);
1188     @(posedge clock);
1189     CoinInserted <= 1;
1190     @(posedge clock);
1191     @(posedge clock);
1192     @(posedge clock);
1193     @(posedge clock);
1194     @(posedge clock);
1195     @(posedge clock);
1196     @(posedge clock);
1197     @(posedge clock);
1198     StartGame <= 0;
1199     @(posedge clock);
1200     StartGame <= 1;
1201     @(posedge clock);
1202     @(posedge clock);
1203     LoadShapeNow <= 0; ShapeLocation <= 2'b10;
1204     @(posedge clock);
1205     @(posedge clock);
```

```
1206      @(posedge clock);
1207      LoadShapeNow <= 1;
1208      @(posedge clock);
1209      @(posedge clock);
1210      @(posedge clock);
1211      @(posedge clock);
1212      LoadShapeNow <= 0; LoadShape <= 3'b110;
1213      @(posedge clock);
1214      @(posedge clock);
1215      @(posedge clock);
1216      LoadShapeNow <= 1;
1217      @(posedge clock);
1218      @(posedge clock);
1219      @(posedge clock);
1220      LoadShapeNow <= 0; LoadShape <= 3'b001; ShapeLocation <= 2'b00;
1221      @(posedge clock);
1222      @(posedge clock);
1223      @(posedge clock);
1224      @(posedge clock);
1225      LoadShapeNow <= 1;
1226      @(posedge clock);
1227      @(posedge clock);
1228      @(posedge clock);
1229      @(posedge clock);
1230      @(posedge clock);
1231      LoadShapeNow <= 0; LoadShape <= 3'b110;
1232      @(posedge clock);
1233      @(posedge clock);
1234      @(posedge clock);
1235      @(posedge clock);
1236      LoadShapeNow <= 1;
1237      @(posedge clock);
1238      @(posedge clock);
1239      @(posedge clock);
1240      @(posedge clock);
1241      @(posedge clock);
1242      LoadShapeNow <= 0; LoadShape <= 3'b101; ShapeLocation <= 2'b11;
1243      @(posedge clock);
1244      @(posedge clock);
1245      @(posedge clock);
1246      @(posedge clock);
1247      LoadShapeNow <= 1;
1248      @(posedge clock);
1249      @(posedge clock);
1250      @(posedge clock);
1251      @(posedge clock);
1252      @(posedge clock);
1253       LoadShapeNow <= 0; LoadShape <= 3'b100; ShapeLocation <= 2'b01;
1254      @(posedge clock);
1255      @(posedge clock);
1256      @(posedge clock);
1257      @(posedge clock);
1258      LoadShapeNow <= 1;
1259      @(posedge clock);
1260      @(posedge clock);
1261      @(posedge clock);
1262      @(posedge clock);
1263      @(posedge clock);
1264      GradeIt <= 0;
1265      @(posedge clock);
1266      @(posedge clock);
1267      @(posedge clock);
1268      @(posedge clock);
1269      @(posedge clock);
1270      GradeIt <= 1;
1271      @(posedge clock);
1272      @(posedge clock);
1273      @(posedge clock);
1274      @(posedge clock);
1275      @(posedge clock);
1276      @(posedge clock);
```

```
1277     @(posedge clock);
1278     @(posedge clock);
1279     @(posedge clock);
1280     GradeIt <= 0;
1281     @(posedge clock);
1282     @(posedge clock);
1283     @(posedge clock);
1284     @(posedge clock);
1285     GradeIt <= 1;
1286     @(posedge clock);
1287     @(posedge clock);
1288     @(posedge clock);
1289     @(posedge clock);
1290     @(posedge clock);
1291     @(posedge clock);
1292     @(posedge clock);
1293     @(posedge clock);
1294     @(posedge clock);
1295     @(posedge clock);
1296     @(posedge clock);
1297     @(posedge clock);
1298     GradeIt <= 0; Guess <= 12'b101_011_001_110;
1299     @(posedge clock);
1300     @(posedge clock);
1301     @(posedge clock);
1302     @(posedge clock);
1303     @(posedge clock);
1304     @(posedge clock);
1305     @(posedge clock);
1306     @(posedge clock);
1307     @(posedge clock);
1308     @(posedge clock);
1309     GradeIt <= 1;
1310     @(posedge clock);
1311     @(posedge clock);
1312     @(posedge clock);
1313     @(posedge clock);
1314     @(posedge clock);
1315     @(posedge clock);
1316     GradeIt <= 0;
1317     @(posedge clock);
1318     @(posedge clock);
1319     @(posedge clock);
1320     @(posedge clock);
1321     @(posedge clock);
1322     @(posedge clock);
1323     @(posedge clock);
1324     @(posedge clock);
1325     @(posedge clock);
1326     @(posedge clock);
1327     GradeIt <= 1;
1328     @(posedge clock);
1329     @(posedge clock);
1330     @(posedge clock);
1331     @(posedge clock);
1332     @(posedge clock);
1333     @(posedge clock);
1334     GradeIt <= 0; Guess <= 12'b101_011_001_110;
1335     @(posedge clock);
1336     @(posedge clock);
1337     @(posedge clock);
1338     @(posedge clock);
1339     @(posedge clock);
1340     @(posedge clock);
1341     @(posedge clock);
1342     @(posedge clock);
1343     @(posedge clock);
1344     @(posedge clock);
1345     GradeIt <= 1;
1346     @(posedge clock);
1347     @(posedge clock);
```

```
1348     @(posedge clock);
1349     @(posedge clock);
1350     @(posedge clock);
1351     @(posedge clock);
1352     GradeIt <= 0; Guess <= 12'b101_011_001_110;
1353     @(posedge clock);
1354     @(posedge clock);
1355     @(posedge clock);
1356     @(posedge clock);
1357     @(posedge clock);
1358     @(posedge clock);
1359     @(posedge clock);
1360     @(posedge clock);
1361     @(posedge clock);
1362     @(posedge clock);
1363     GradeIt <= 1;
1364     @(posedge clock);
1365     @(posedge clock);
1366     @(posedge clock);
1367     @(posedge clock);
1368     @(posedge clock);
1369     @(posedge clock);
1370     GradeIt <= 0; Guess <= 12'b101_011_001_110;
1371     @(posedge clock);
1372     @(posedge clock);
1373     @(posedge clock);
1374     @(posedge clock);
1375     @(posedge clock);
1376     @(posedge clock);
1377     @(posedge clock);
1378     @(posedge clock);
1379     @(posedge clock);
1380     @(posedge clock);
1381     GradeIt <= 1;
1382     @(posedge clock);
1383     @(posedge clock);
1384     @(posedge clock);
1385     @(posedge clock);
1386     @(posedge clock);
1387     @(posedge clock);
1388     GradeIt <= 0; Guess <= 12'b101_011_001_110;
1389     @(posedge clock);
1390     @(posedge clock);
1391     @(posedge clock);
1392     @(posedge clock);
1393     @(posedge clock);
1394     @(posedge clock);
1395     @(posedge clock);
1396     @(posedge clock);
1397     @(posedge clock);
1398     @(posedge clock);
1399     GradeIt <= 1;
1400     @(posedge clock);
1401     @(posedge clock);
1402     @(posedge clock);
1403     @(posedge clock);
1404     @(posedge clock);
1405     @(posedge clock);
1406     @(posedge clock);
1407     @(posedge clock);
1408     @(posedge clock);
1409     GradeIt <= 0; Guess <= 12'b101_110_100_001;
1410     @(posedge clock);
1411     @(posedge clock);
1412     @(posedge clock);
1413     @(posedge clock);
1414     @(posedge clock);
1415     @(posedge clock);
1416     @(posedge clock);
1417     @(posedge clock);
1418     @(posedge clock);
```

```
1419      @(posedge clock);
1420      GradeIt <= 1;
1421      @(posedge clock);
1422      @(posedge clock);
1423      @(posedge clock);
1424      @(posedge clock);
1425      @(posedge clock);
1426      @(posedge clock);
1427      @(posedge clock);
1428      @(posedge clock);
1429      @(posedge clock);
1430      @(posedge clock);
1431      @(posedge clock);
1432      #1 $finish;
1433    end
1434
1435 endmodule: ZorgGame_test
```